Passive NFS Tracing of Email and Research Workloads

Daniel Ellard, Jonathan Ledlie, Pia Malkani, Margo Seltzer
{*ellard,jonathan,malkani,margo*}*@eecs.harvard.edu*
Division of Engineering and Applied Sciences, Harvard University

Abstract

We present an analysis of a pair of NFS traces of contemporary email and research workloads. We show that although the research workload resembles previously-studied workloads, the email workload is quite different. We also perform several new analyses that demonstrate the periodic nature of file system activity, the effect of out-of-order NFS calls, and the strong relationship between the name of a file and its size, lifetime, and access pattern.

1 Introduction

Trace-based analyses have guided and motivated contemporary file system design for the past two decades. The original analysis of the 4.2BSD file system [9] motivated many of the design decisions of the log-structured file system (LFS) [13]. The revisitation of the original BSD study [1] confirmed the community's earlier results and further drove file system design and evaluation towards the support of distributed file systems as well as computer science engineering and research workloads. In the late 1990's, our repertoire of trace-based studies was expanded to include the increasingly dominant desktop systems of Microsoft [12, 15] and new workloads such as WWW servers [12]. It is clear from the literature of trace-based studies that there are many interesting and important workloads to consider when designing a file system, and that new workloads emerge as new applications and uses for file systems appear. We believe that as the community of computer users has expanded and evolved there has been a fundamental change in the workloads seen by file servers, and that the research community must find ways to observe and measure these new workloads.

In this paper, we present an analysis of two contemporary NFS workloads: EECS, a CS research workload, and CAMPUS, the central computing facility for our university. Both of these systems exhibit new characteristics that future file system designs might be able to exploit.

The EECS workload extends the pattern of research workloads of earlier studies, but ventures into new territory – the EECS workload is dominated by metadata requests and has a read/write ratio of less than 1.0.

The CAMPUS workload is almost entirely email, and is dominated by reading. Nearly all of the active files on CAMPUS fall into one of four categories – mailbox files, lock files, scratch files used by mail clients, and configuration files. Each category of file has a predictable size, lifespan, and access pattern. Virtually all files on CAMPUS can be correctly categorized by their filename, and therefore filenames might provide useful hints to the file system about each file.

The contributions of this work are:

- A large set of anonymized traces from both technical/research and email workloads
- An analysis of these new traces and comparison to prior traces
- New techniques for analyzing NFS traces, including a new method for quantifying workload sequentiality and understanding its impact on file servers
- Evidence that many properties of a file are predicted by its name
- Tools to gather new anonymized NFS traces

The rest of this paper is organized as follows. Section 2 describes our tracing software and the trace anonymization process. In Section 3, we give an overview of the systems we studied and the traces we gathered. We discuss the advantages and challenges of gathering file system traces from NFS environments in Section 4. In Section 5, we describe our new traces and compare them to the results of previous trace-based studies, highlighting the similarities and differences. In Section 6, we discuss the new analyses and findings revealed by our traces, and in Section 7, we summarize our findings and discuss directions for future research.

2 Our Tracing Software

Our traces were collected by attaching a computer running our NFS packet snooping software (based on an extensively modified version of `tcpdump`) to a mirror port on the network switch hosting the servers under study. Our tracing software can handle any combination of NFSv2 and NFSv3, TCP or UDP transport, gigabit

Ethernet, and jumbo frames. Unlike other tracing tools, we also support some forms of TCP packet coalescing and automatic anonymization of the traces. Our software is also open-source, easy to configure, runs unattended, and is portable across different flavors of UNIX.

The output of our tracing software is either a time-stamped literal record for each NFS call and response observed over the network or an anonymized version of these records. The anonymization process replaces all UIDs, GIDs, and IP addresses in the traces with arbitrary but consistent values. Filenames and paths are anonymized by pathname components: directories are anonymized individually, so that if two paths have a common prefix path, their anonymized forms will share a common prefix as well. In addition, filename suffixes are anonymized separately from the rest of the filename, so all files that share the same suffix will have anonymized names that end in the anonymized form of that suffix.

The anonymization process is configurable – the mapping for the anonymization of any value can be overridden. For example, in the anonymization of our own data, we disable the anonymization of many common file and directory names (such as `CVS`, `.inbox` and `.pinerc`) or components (such as `lock`) and specific UIDs (such as `root` and `daemon`). We also treat several suffixes and prefixes (such as `#`, `,v` and `~`) specially, to preserve the relationship between the anonymized form of filenames containing these suffixes or prefixes and the anonymized form of the filenames without them. It is also possible to configure the anonymizer to omit all filename, UID, GID, and IP information entirely. This makes some analyses impossible, but still provides a great deal of useful information to file system researchers.

Our anonymization approach is vulnerable to a known-text attack by an adversary who has detailed knowledge about the system being traced. For example, an attacker who knows exactly when a particular user accessed a particular file can find the user's anonymized UID and GID and the anonymized filename. Similarly, a chosen-text attack on the system while it is being traced can be used to find the anonymized form of any filename that the attacker can create. Both of these attacks require direct access to or real-time information about the system being traced. An insider can exploit his or her ability to use the file system to gain knowledge about the other users of the system, but an outsider who does not have this leverage cannot reverse the anonymization process. We do not use hashing or any other deterministic method to do the anonymization, because that would allow an attacker to perform a known-text attack without access to the traced system, and it would also allow direct comparison of filenames and other information between traces gathered at different sites.

3 Traced Systems

In order to examine new workloads and place our work in the context of previous studies, we gathered two sets of traces. The EECS trace is reminiscent of the frequently studied computer science departmental workload. The CAMPUS workload is from our university's central computing center and is almost entirely email. The two workloads are summarized in Table 1.

3.1 The EECS System

The main EECS NFS server is a Network Appliance Filer that serves as the primary home directory server for our computer science department. The workload is a mix of research, software development, and course work. EECS users can directly mount their home directories onto their workstations via NFS, Samba, or other protocols. In the EECS system, all non-NFS file access protocols are routed through an intermediate host, which converts these protocols to NFS but prevents us from directly identifying the source of the resulting activity.

Although there is no standard EECS client, a typical client is a UNIX or Windows NT machine with more than 128MB RAM and local copies of system software and most utilities. Most of the EECS clients use NFSv3, but many use NFSv2. All EECS clients use UDP to communicate with the server. The EECS server is used primarily for home directories and shared project and data files. Unlike what we will see in the CAMPUS workload, the EECS traces do not contain any email traffic. In the EECS system, email and WWW service is provided by other servers, and user inboxes are stored on a separate server.

There are no user quotas on the EECS system. The aggregate disk capacity of the EECS client machines is much larger than the capacity of the EECS server. Much of the research data used in the EECS environment is stored on other servers; the only items stored on the EECS server are home directories, data that need to be shared, or data that users want to have backed up. The traces do not include any backup activity.

3.2 The CAMPUS System

CAMPUS is a collection of machines serving the computing needs for the bulk of the administration, college, and graduate school of the University. It handles email and web service for the majority of the students, faculty, and administrators, and has approximately 10,000 active user accounts.

CAMPUS storage is distributed over three NFS servers hosting a total of fourteen 53GB disk arrays. Each NFS server has several network interfaces and they

CAMPUS	EECS
Storage for the campus SMTP, POP and login servers	Storage for the EECS department home directories
Most NFS calls are for data	Most NFS calls are for metadata
Reads outnumber writes by a factor of 3.0	Writes outnumber reads by a factor of 1.4
Peak load periods highly correlated with day of week and time of day	Unpredictable interactive load, but predictable background activity
20% of the files accessed, and 95+% of the data read and written comes from mailboxes	No mailboxes, some mail lock and temporary files
50% of files accessed are mailbox locks	A large number of locks for mail and other applications
Most blocks live for at least ten minutes	Most blocks die in less than one second
Almost all blocks die due to overwriting	Blocks die due to a mix of overwriting and file deletion

Table 1: Characteristics of CAMPUS and EECS.

are configured, via `gated`, to appear as fourteen virtual hosts, one per disk array. Because all NFS traffic to a particular disk array uses an IP address unique to that disk array, we can monitor traffic to the individual arrays.

The connection between the CAMPUS clients and servers is a gigabit Ethernet using jumbo (9000 byte) frames. All clients use the NFSv3 over TCP.

Each of the fourteen disk arrays contains the home directories for a subset of the CAMPUS users. Users are given a default quota of 50MB for their home directories. Users are distributed among the disk arrays according to the first letters of their login names. We gathered long-term traces for two arrays and short-term traces for seven others. We computed summary statistics and general usage patterns for all nine of the traced arrays and found them to be similar. We chose to use the array named home02 for our in-depth analysis.

The central email, WWW, general login, and CS course servers mount the disk arrays via different networks. Our traces capture only the NFS traffic between the email and general login servers and the disk arrays; we do not capture the NFS traffic generated by serving personal home pages or by students working on CS assignments. Statistics provided by CAMPUS sysadmins confirm that the subnet that carries the email and general-purpose login traffic represents the vast majority of the total CAMPUS traffic. The CAMPUS traces do not include backup activity.

Unlike many other systems, mail spools are not kept in dedicated partitions; users' inboxes are located inside their home directories. However, the activity of the CAMPUS system is so dominated by email that in some sense the file systems used for home directories can be considered to be dedicated email partitions. Most users of the CAMPUS system access mail remotely via a POP or SMTP server from their PC or Macintosh. Our traces do not include this POP and SMTP mail delivery activity as such, but do contain records of the NFS traffic this activity generates.

3.3 Data Used

We gathered several months of traces on both systems and then computed summary statistics for the entire trace period and additional analyses for the months of 10/2001 and 11/2001. We observed that the workload for CAMPUS is quite consistent over the entire trace period (once the school schedule, including weekends and holidays, is taken into account). The EECS workload shows considerably more variation from day to day and week to week, but the variance falls within reasonable limits.

For the analyses in this paper, we selected the one-week period of October 21 through October 27, 2001. This week was chosen because it is "typical" in the sense that it is in the middle of the semester and contains no holidays or other unusual events, and because our data for this week contain no gaps. For the EECS system, although each week varies from the others, and none of the weeks appear "average", this week is no more atypical than any other. Each table and graph is labeled with the subset of the data used.

Table 2 shows the summary statistics for the average daily activity of our traces, for both a three month subset and the one week subset used for our in-depth analyses, and compares them to the same statistics from the Baker and Roselli traces [1, 12]. Note that the RES, INS, and NT traces are kernel-level traces of local file systems, and do not show the effect of client-side caching. The Sprite traces, on the other hand, use a different form of client-side caching than NFS.

CAMPUS is an order of magnitude busier than any of the other systems, particularly in terms of the amount of data read and written.

4 Trace Analysis via NFS

Passive NFS tracing is an old idea, and has been used in many trace studies and file system experiments [2, 3, 5, 7, 8]. The technique of passive tracing on a broadcast network evolved in parallel with broadcast net-

	CAMPUS 9/1-11/30	EECS 9/1-11/30	CAMPUS 10/21-10/27	EECS 10/21-10/27	INS	RES	NT	Sprite
Year of Trace	2001	2001	2001	2001	2000	2000	2000	1991
Days	91	91	7	7	31	31	31	8
Total ops (millions)	29.9	2.30	26.7	4.44	8.30	3.20	3.87	0.432
Data read (GB)	135.7	2.27	119.6	5.10	3.05	1.70	4.04	5.36
Read ops (millions)	19.29	0.35	17.29	0.461	2.32	0.303	1.27	0.207
Data written (GB)	45.0	2.94	44.57	9.086	0.542	0.455	0.639	1.16
Write ops (millions)	5.93	0.438	5.73	0.667	0.15	0.071	0.231	0.057
Read/Write bytes ratio	3.01	0.77	2.68	0.56	5.6	3.7	6.3	4.6
Read/Write ops ratio	3.25	0.80	3.01	0.69	15.4	4.27	4.49	3.61

Table 2: A summary of average daily activity during the trace periods. The subset from 10/21 through 10/27 is used for most of our analyses. The INS, RES, NT, and Sprite numbers are from the Roselli and Baker trace studies. INS is an instructional workload, RES is a research workload, and NT is a Windows NT desktop workload.

works such as Ethernet and client/server protocols such as RPC. NFS trace studies began to appear in the literature soon after NFS clients appeared on computer science department networks [7], but they received relatively little attention compared to kernel-based trace studies or studies of other distributed file systems. This changed, however, as NFS became ubiquitous and better tracing tools and methodologies for analyzing NFS traces were devised [2, 8].

Passive NFS tracing is attractive from a research perspective because NFS is a portable and widespread protocol, used in a broad variety of real-world contexts, and so its analysis is applicable to many interesting real-world problems. Unfortunately, we have found that even while computing has become pervasive, it has become increasingly difficult to gather meaningful traces of production servers outside of research labs.

The primary difficulty in gathering traces from production servers is a combination of social, ethical, and legal issues. Detailed file system traces can reveal sensitive information about the activities of organizations or individual users, and the administrators of these systems have justifiable concerns about protecting the privacy of their users. During our data collection, we contacted several commercial ISPs in an effort to gather traces from their sites, and although several of the administrators of these ISPs expressed interest, none of them would permit us to gather traces because of privacy concerns. To address this concern, we added an anonymization step to our tracing procedure, as described in Section 2. This step transforms the traces in such a way that user-specific information such as UIDs or filenames is not revealed, while preserving the information necessary for almost any analysis. We hope that the existence of this anonymizer and the value of our findings will convince ISPs to permit tracing, allow these traces to be shared among the research community, and thereby invigorate contemporary file system design.

A second obstacle to gathering traces is that system administrators are understandably unwilling to modify their systems to allow direct instrumentation because of the risk associated with loading new, non-production, and potentially buggy kernel patches into their critical systems as well as the concern about the additional load it might place on their systems. Passive NFS tracing addresses this concern perfectly; capturing and recording NFS packets from the network requires no changes to the clients or servers and does not introduce any new load on the system.

Despite the decline in the number of new NFS trace studies in recent years, we believe that the technique of passive NFS tracing in tandem with trace anonymization is more important now than ever, because it may allow researchers to trace systems that would otherwise be completely inaccessible.

4.1 Difficulties of Analyzing NFS Traces

As mentioned in earlier NFS trace studies, the analysis of passive NFS traces presents several challenges, particularly when trying to relate these analyses to those done on kernel-based traces [8]. This subsection provides an overview of the challenges of comparing NFS traces with kernel-based traces:

- Details of the underlying file system are hidden.
- The NFS interface is different from the canonical FS interface.
- Client-side caching skews the observed workload.
- Some NFS calls and responses are lost.
- NFS calls may be reordered *en route*.

Just as the idea of passive NFS tracing is not new, methods for dealing with most of these problems are not new, and have been described at length elsewhere [2, 5, 7, 8], and we will discuss these methods only briefly. The problem of reordered calls, however, is something we have not seen addressed in the literature, and so we will discuss our approach in detail in Section 4.2.

4.1.1 The Underlying File System is Hidden

Much information about the underlying server file system is not revealed by any analysis of the messages between NFS clients and servers. For example, it is impossible to learn much about files or directories that are never accessed via NFS, or any information about the actual on-disk layout of the files and directories. It is also impossible to reconstruct the complete file system hierarchy *a priori* – but as shown in earlier studies [2, 8], it is possible to reconstruct the active parts of the hierarchy on-the-fly by learning the relationship between directories and their contents as revealed by `lookup` calls and responses. This method is effective for our traces; after processing several minutes of traces, the probability is very small that we will encounter a file or directory whose parent directory has not already been seen.

Another problem with tracing at this level is that little information about the internal state of the server (such as the state and contents of its cache) is revealed. Because we are interested in characterizing the server workload, which is independent of the internal state of the NFS server, this does not impede our analyses.

4.1.2 The NFS Interface

Analyses that depend on specific details of the file system interface cannot be done on NFS traces. For example, some analyses require knowledge of file `open` and `close` system calls, but neither of these calls exists in NFS. Methods for augmenting a trace with virtual `open` and `close` events are described in related work [2, 8]. We describe our approach for finding *runs* in Section 4.2.

4.1.3 The Effect of Client-Side Caching

The effect of client-side caching is an obstacle to inferring the actual client workload from the workload observed by an NFS server. In order to reduce the latency of client operations, NFS allows clients to cache data and metadata in a weakly consistent manner. This means that some client operations are absorbed by the client cache and never reach the server, or are observed only as `getattr` calls when the client checks whether the data in its cache is still valid.

The strategies employed by some clients can also have the effect that some I/O operations seen by the server might not correspond to actual calls by the client application, but instead are the effect of the client-side operating system performing read-ahead in an attempt to warm its cache with data it anticipates will be requested soon.

The effect of client-side caching and methods to infer actual client workload from the server traffic have been studied extensively [2, 3, 5], but still present new challenges in heterogeneous networks because NFS allows considerable flexibility in how caching is implemented and in the degree of consistency provided to different observers of the file system.

Because our research is primarily concerned with server workloads, which are strongly shaped by client-side caching, we do not want to remove the effects of client-side caching and therefore make no attempt to control for them. This implies that direct comparisons of operation counts and the volume of data read and written between our NFS traces and traces from local file systems should show that the NFS traces contain relatively more metadata traffic (to validate the cache contents) but less data actually read (if the cache is often valid).

4.1.4 Lost NFS Calls/Responses

On the CAMPUS system our monitor consisted of a single gigabit Ethernet port on a fully-switched gigabit network. During bursts of heavy activity, the monitor port simply did not have the bandwidth to forward all of the network traffic. This problem was compounded by the fact that it is impossible to decode an NFS response without seeing the call, so losing a call effectively results in losing both the call and its response. By analyzing the call stream for unexpected holes, and by counting the number of call and responses messages that had no corresponding response or call, we estimate that during period of heavy activity, we lost as many as 10% of the packets through the switch, and for short bursts the percentage could have been even higher. In contrast, on EECS the monitor port was as fast as the port to the server, so we did not observe this problem.

4.1.5 Reordered NFS Calls

Some analyses, such as determining whether a set of accesses are sequential or random, are sensitive to the order of calls. Out-of-order calls occur when NFS calls are delivered to the server in a different order than they were issued by the application. This reordering is largely an artifact of the conventional NFS architecture, in which separate processes, called `nfsiods`, issue the actual network calls. Although a client's calls are dispatched to the `nfsiods` in order, the process scheduler determines the order in which the `nfsiods` run.

To confirm and measure this effect, we performed an experiment using our own clients and server on an isolated network. When the client ran only one `nfsiod`, no call reorderings occurred, but as additional `nfsiods` were added, call reordering became more frequent. In the most extreme case as many as 10% of the packets were reordered, and some calls were delayed by as much as 1 second, although no network errors or packet losses were observed. This effect is more common when UDP is used as the RPC transport, but can also occur with TCP.

4.2 Detecting Runs in NFS

Earlier trace studies [1, 9, 12, 15] describe workloads in terms of sequential runs. The notion of a sequential run is important to the heuristics that file systems use to efficiently process a stream of requests to a file. For example, FFS prefetches blocks when an access pattern appears sequential and does not when it appears non-sequential. Determining a client's underlying access pattern is an important part of optimizing file system access.

In order to compare our traces to earlier studies, we needed a method to divide NFS records into runs on a file. When we applied the conventional analysis of considering all consecutive accesses to a single file as a sequential run, we obtained a much higher percentage of random accesses than previous work had led us to expect. We determined that the apparent randomness was due to two factors: the reordering of requests introduced by multiple client-side `nfsiod` processes and the omission of a small percentage of NFS records from our logs. Because of these factors, the conventional methods are ineffective for NFS analysis.

Previous studies define a *run* as the series of accesses to a file between each `open` and subsequent `close`, and an *access* as either a single read or write. Because `open` and `close` do not exist in NFS, we use the following methodology to create runs:

1. We associate a list of accesses with each file, adding a record to this list whenever we see a `read` or `write` to the file in the trace.
2. We then convert this list into a collection of one or more runs.
   (a) If the last item referenced the end-of-file, begin a new run.
   (b) If the last item in a list is old (*e.g.,* older than 30 seconds) begin a new run.
   (c) Else, add the current item to the current run.

Using this mechanism, a series of accesses can be split into one or more runs. We also evaluated other methods to break lists of accesses into runs. For example, a break might occur when a gap of several seconds occurs between two accesses or when a large backwards seek occurs. We found that these other methods produced inconsistent results.

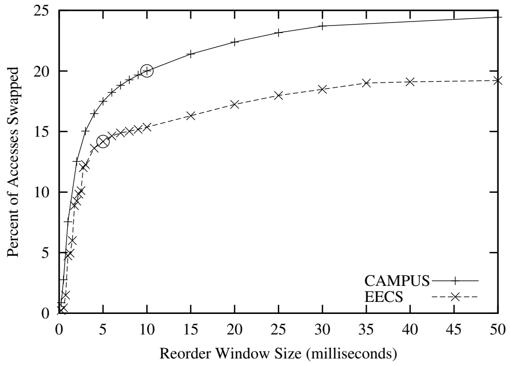

Figure 1: The effect of different window sizes on the percentage of swapped accesses. The sorting window size chosen for each host is shown with a circle: for EECS, 5ms is sufficient, but for CAMPUS 10ms is better.

After splitting accesses into runs, we separately determined each run's access pattern. Informally, a run is sequential if each request begins where the previous one left off, for all requests. In NFS terms, a run is sequential if, for all accesses in the run, $\text{offset}_i = \text{offset}_{i-1} + \text{count}_{i-1}$, where i is the index of the current access in a run and count_{i-1} is the number of bytes accessed in the previous access. Offsets and counts are rounded up to blocksizes of 8k. For example if a series of offset(counts) was 0k(8k), 8k(8k), 16k(7k), 24k(8k), we would consider this series to be sequential despite the missing 1k between the third and fourth requests. If a run has the qualities of being sequential and also accesses the file from offset 0 through to *eof*, it is called *entire*. If it is not sequential, the run is called *random*. A run is called "read" if it contains only read requests, "write" if is contains only writes, and "read-write" if it contains one or more of each.

If we do nothing to compensate for the reordering that occurs due to `nfsiod` scheduling, we observe an unnaturally large percentage of random accesses. In order to avoid this phenomenon, we partially sort requests in ascending order within a small temporal window, which we call a *reorder window*. For each request, we look ahead t milliseconds to see if a nearby request should be swapped with the current request, and swap them if they are out of order. We show the results of using differing *reorder window* sizes on a subset of the data (Wednesday 10/24/2001, 9am-12pm) in Figure 1.

Note that we want to use the minimum sorting window that removes the reordering effect of the `nfsiods` but does not mask true client randomness – with an infinite sorting window, any workload that visits every block of a file in any order will appear sequential. Both workloads show a great improvement in observed sequentiality with a sorting window of only a few milliseconds and then exhibit a knee, after which larger sorting windows give much smaller gains in the observed sequentiality. Empirically, the results from this analysis suggested that we should use a window size of $t = 5$ milliseconds for EECS and $t = 10$ milliseconds for CAMPUS. For the remainder of this paper, the sequentiality analysis includes this sorting step, unless otherwise noted.

5 Comparison With Previous Traces

In this section, we place our traces in the context of several often-cited traces from earlier studies. We examine run patterns and block and file lifetimes. In Section 6, we discuss analyses specific to our traces.

5.1 Run Patterns

Most previous studies categorize workloads by their level of sequentiality, because many file system layout optimizations rely upon some degree of sequentiality.

The rightmost columns of Table 3 compare EECS and CAMPUS to three historical traces, using the heuristic described in Section 4.2 to find the runs in our traces. Both EECS and CAMPUS have significantly different run patterns than earlier traces. Both workloads contain a much higher percentage of write runs (especially EECS, where write operations already outnumber read operations in the overall operation count). We initially believed that much of the EECS write activity is due to late night batch jobs, but examining only peak hours revealed essentially the same percentages. Peak hour analysis for CAMPUS also yields the same percentages shown in Table 3. In EECS, we believe that the dominance of write traffic is because most client machines are used by one user and that user's files experience little cache invalidation. In CAMPUS, the read-write imbalance has a different cause: most writes are short appends to mailbox files, and most reads are long reads of mailbox and other mail-related files.

Note that the classification of "random" and "sequential" used in Table 3 follows the heuristics used by many file systems, which categorize all non-sequential access patterns as random. Therefore highly regular access patterns, such as stride access patterns or reverse scans, would be overlooked by this classification. A visual inspection of the non-sequential access patterns in our traces did not reveal a significant number of accesses that had any discernible pattern other than sequential sub-accesses separated by seeks.

Figure 2 illustrates the file size-based access patterns of EECS and CAMPUS. EECS is similar to previously

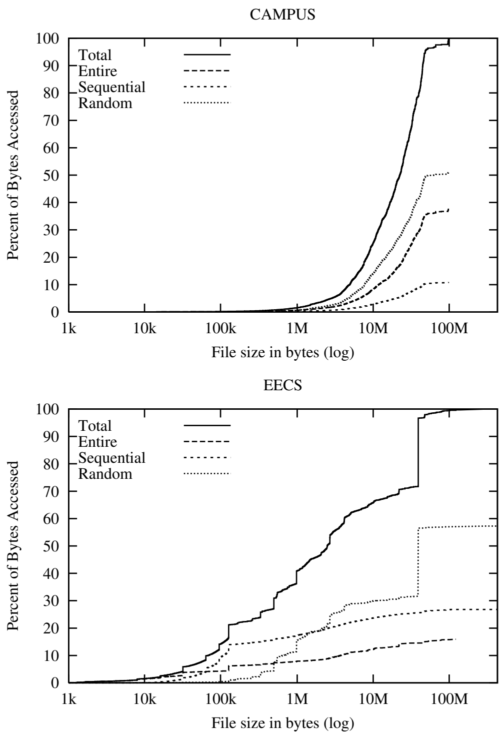

Figure 2: Percentage of bytes accessed randomly, sequentially, or in their entirety verses the total bytes accessed. Each run is categorized according to its access pattern and then all of the bytes accessed in this run are added to the subtotal for this category. These runs are computed by the heuristic described in Section 4.2.

analyzed file systems where a large percentage of accesses come from files smaller than 1M. Almost 60% of bytes are accessed randomly, which is about the same as Roselli's NT workload but much larger than most other workloads. Again like the NT workload, long files read in their entirety constitute 30% of the total in EECS. CAMPUS is similarly dominated by random and entire runs, although as discussed in Section 6.4 most of these runs that are categorized as "random" do contain long, completely sequential sub-runs, and should be considered highly sequential.

The vast majority of CAMPUS bytes transferred come from files larger than 1M. This is unlike almost all of the previous work on run analysis, except for the two outlier traces of the Baker study [1].

Previous analysis, Roselli in particular, showed that the majority of bytes from files over 100k are accessed randomly. As mentioned in Section 4.2, our data indicates that "random" is too strong a term and too coarse a measurement. In our traces, most runs that would be categorized as random in the entire/sequential/random

Access Pattern	CAMPUS	EECS	NT	Sprite	BSD	CAMPUS	EECS
	Raw					Processed	
Reads (% total)	53.1	16.6	73.8	83.5	64.5	53.1	16.5
Entire (% read)	47.7	53.9	64.6	72.5	67.1	57.6	57.2
Sequential (% read)	29.3	36.8	7.1	25.4	24.0	33.9	39.0
Random (% read)	23.0	9.3	28.3	2.1	8.9	8.6	3.8
Writes (% total)	43.8	82.3	23.5	15.4	27.5	43.9	82.3
Entire (% write)	37.2	19.6	41.6	67.0	82.5	37.8	19.6
Sequential (% write)	52.3	76.2	57.1	28.9	17.2	53.2	78.3
Random (% write)	10.5	4.1	1.3	4.0	0.3	9.0	2.1
Read-Write (% total)	3.1	1.1	2.7	1.1	7.9	3.0	1.1
Entire (% r-w)	1.4	4.4	15.9	0.1	NA	3.5	5.8
Sequential (% r-w)	0.9	1.8	0.3	0.0	NA	2.1	7.3
Random (% r-w)	97.8	93.9	83.8	99.9	75.1	94.3	86.8

Table 3: File access patterns using entire/sequential/random categorization. The leftmost two columns show CAMPUS and EECS traces divided into runs according to the sorting mechanism described in Section 4.2, but otherwise unaltered. The rightmost two columns show the CAMPUS and EECS traces divided into runs using the complete methodology presented in Section 4.2. to account for request reordering produced by `nfsiods` and prevent small seeks (of less than 10 8k blocks) from changing a run from sequential to random. In both this presentation and that in section 6.4, singleton runs are either sequential (if they access only part of the file) or entire (if they access the entire file). The third, fourth, and fifth columns show the results from Roselli's NT, the Sprite, and the BSD studies, respectively.

taxonomy are actually composed primarily of sequential sub-runs separated by short seeks. We present a new metric for run sequentiality in Section 6.4.

5.2 File and Block Lifetimes

The distribution of block and file lifetimes is a workload characterization that can be exploited by the file system. If blocks do not live long, it may be possible to keep them in memory and avoid ever writing them to disk. If the block lifetimes are bimodal, then once a block lives sufficiently long, it might be beneficial to optimize its on-disk layout under the assumption that the block will live for a long time and the cost of reorganization can be amortized across many future accesses.

We used Roselli's "create-based" method [12] to calculate block lifetime statistics. We processed our data in two separate phases. In Phase 1, we record both block births and deaths. In Phase 2, the *end margin* of Roselli, we record only block deaths. To remove sampling bias for blocks born early in the first phase, we remove any death records for blocks with lifespans longer than the length of the second phase. All blocks whose lifetimes exceed the length of Phase 2 are considered *end surplus*.

5.2.1 Analysis

We ran a set of five 24-hour block life analyses on CAMPUS and EECS for the weekdays in our trace period. The first phase of each test began at 9am and ran for 24 hours with a 24-hour end margin. We chose a 24-hour end margin because shorter margins did not capture the relatively high percentage of deaths that occur between 18 hours and 24 hours. Longer tests agree with earlier observations that a block that lives for a day is likely to live for a relatively long time. The daily end surplus ranges between 2.1% and 5.9% for CAMPUS and between 3.5% and 9.5% for EECS. Logically, this daily surplus must eventually be balanced by deletions (via periodic purges when disk quotas are reached or the file system fills up, or more gradual and steady deletion), because otherwise CAMPUS and EECS would be filled in a matter of weeks. We have not investigated this phenomenon, however.

5.2.2 Summary of Block Births and Deaths

The summary statistics for block deaths and births are shown in Table 4. Both CAMPUS and EECS exhibit similar trends with respect to block births. In both systems, most births are due to actual data writes, as opposed to preallocation of space (*e.g.,* using `lseek` to lengthen a file). In fact, the number of file extensions is mildly exaggerated, because writes that follow an `lseek` past the end-of-file are interpreted as extension writes to all the newly created blocks; that is, not only the blocks explicitly written, but all unwritten blocks between the previous end-of-file and the new block are counted as extensions. In the future it might be useful

	CAMPUS	EECS
Total Births (millions)	28.4	9.8
Due to Writes	99.9 %	75.5 %
Due to Extension	$\ll$ 0.1 %	24.5 %
Total Deaths (millions)	27.5	9.2
Due to Overwrites	99.1 %	42.4 %
Due to Truncates	0.6 %	5.8 %
Due to File Deletion	0.3 %	51.8 %

Table 4: Daily block life statistics for 10/22-10/26/2001. Note that only the block deaths that occur within 24 hours of the block birth are reported here, and so the total number of blocks that are born and die within this five day period is higher than the **Total Deaths** figure.

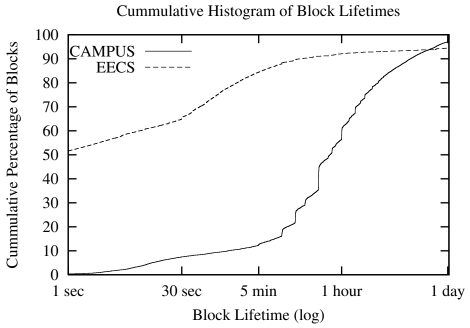

Figure 3: The cumulative distribution of block lifetimes for each day 10/22-10/26/2001.

to distinguish between these cases. Even with this overcounting, however, relatively few blocks are created via file extensions, particularly on CAMPUS.

Earlier studies found that most blocks die due to overwriting. In the Roselli trace results [12], the percentage of blocks that die due to overwriting is always greater than 50% and often between 85%-99%. Our results follow this pattern, but show an important difference between CAMPUS and EECS. On CAMPUS, more than 99% of the blocks die due to overwrites. For CAMPUS almost all the bytes written are to mailboxes, which are never deleted but are overwritten frequently.

The distribution is more varied on EECS. There are considerably fewer overwrites on EECS than CAMPUS, and many more removes. This is due to the research-oriented workload on EECS: tasks such as compiling programs, using source control tools, and manipulating data can create and delete many temporary files. There are also approximately 10,000 deletes per day of small files with names of the form "`Applet_*_Extern`", which are files created by UNIX window managers. Web browser caches and email composition files make up most of the rest of the deleted files.

On CAMPUS, more than 96% of the files created and deleted during the course of a day are zero-length lock files. For EECS, lock files account for only 8% of the files created and deleted each day. Since these are zero-length files, however, they do not consume any data blocks, and so this does not have an effect on the block lifetime statistics.

5.2.3 The Lifespan of Blocks

The lifespan of a block is defined as its time-of-death minus its time-of-birth. Figure 3 illustrates that the two systems are quite different. For EECS, over half the blocks die in less than a second and relatively few blocks live for an entire day; this is similar to the results of the NT study by Vogels [15]. Most of the blocks that die in less than one second on EECS belong to log or index files that are written frequently and in an unbuffered manner.

On CAMPUS, blocks tend to live longer; about half live longer than 10-15 minutes. These results are more reminiscent of the results of Baker [1] than they are of the more recent studies by Vogels [15] and Roselli [12], although the curve of our distributions is similar.

On CAMPUS, blocks live longer because many are born due to mail delivery (at the end of the mailbox) or saving the mailbox before exiting an email client, and blocks die when mail messages are removed from the mailbox, a process that most email clients do in a batch operation. Thus we expect many blocks to live for approximately the same length of time as an average email session. Although we do not know this length, our data suggest mail-reading session times typically range between fifteen minutes and an hour.

In comparison with Roselli's block lifetime analysis results [12], we see that our results are quite different. The only similarity we observed was the fact that both our CAMPUS trace and Roselli's RES trace have a knee at approximately the 10-minute mark. However after this 10-minute mark, CAMPUS has a more gradual increase of block lifetimes, whereas for RES the change is abrupt. On CAMPUS, few blocks live for less than a second. Approximately 50% and 20% die within a second for EECS and Roselli traces, respectively. The end surplus, those blocks that live longer than a day, is about 5%-7% for both CAMPUS and EECS. For Roselli, this percentage varies between 10%-30% for non-NT traces and is about 70% for the NT traces.

6 New Findings

In this section, we discuss observations specific to our traces and to the EECS and CAMPUS workloads.

6.1 Characterizations of EECS and CAMPUS

EECS and CAMPUS differ on a fundamental level. EECS is dominated by writing and by metadata queries, particularly client queries about whether their cached copies of the EECS data are still valid. The CAMPUS workload, in contrast, is dominated by reading and writing large files and by creating and deleting lock files.

CAMPUS is utterly dominated by email and the daily rhythms of user activity. EECS is similar to the research workloads already documented in the literature, but has a lower read/write ratio.

We observe that applications frequently use the file system for ephemeral data storage and for locking. These operations can impose a significant load on file systems, especially when these temporary, cache, or lock files are actually hosted on a different machine than the application. Either file system designers should be sure that systems can identify and handle these cases efficiently, or application designers should be encouraged to use more suitable metaphors for locking and temporary storage.

Our results also provide more evidence that, as shown in previous studies, delayed writes can substantially reduce the amount of actual writing done by the file system, because many blocks do not live long enough to be written. This is particularly true for data blocks on EECS. We also believe that this will be true for the metadata blocks on CAMPUS because of the incessant creation and deletion of short-lived zero-length lock files.

6.1.1 The EECS Workload - Research

The EECS workload is predominantly file attribute calls (`lookup`, `getattr`, and `access`). Most of these calls occur because clients are simply checking to see whether a file has been updated or whether they can use a cached copy. Unlike CAMPUS, which is read-oriented, the overall EECS read/write ratio is 0.69, but varies widely over time, with periods of heavy read activity. This is illustrated in Figure 4. Note that the average hourly read/write ratio shown in Table 5 is skewed by periods of relative inactivity that are dominated by reading, and hence the hourly average is quite different from the overall average.

Somewhat perversely, much of the EECS workload is caching web pages viewed by users running on client workstations. By default, these browser caches are created in a subdirectory of the user's home directory, so they are "cached" on the central file server instead of locally on each machine. Similarly, many of the most frequently created and deleted files on EECS are files created by the window managers and desktop applications of some users (for example, `Applet` files created by GNOME).

Aside from the central storage of the web pages and `Applet` files, if our EECS workload is an instance of the "typical departmental server," then not much has changed in the past fifteen years. Our traffic patterns resemble the model postulated by Ousterhout *et al.*[9], which predicted that as cache sizes grow, most reads will be serviced from cache and write performance will become the bottleneck. This prediction was the motivation for several new developments in file system design, such as LFS [13]. Ousterhout's conjecture that we may eventually be able to replace rewritable media such as magnetic disks with less expensive and write-only storage devices (tested successfully for the Venti system [11]) does not appear to hold for EECS or CAMPUS because even though cache sizes have increased since 1985, the size of file systems and the amount of data accessed by applications has also grown.

We speculate that the NFSv4 lease and delegation mechanisms [10] could eliminate a large fraction of the NFS calls generated by the EECS workload by removing many of the situations where a client is contacting the server simply to confirm that its cached copy of a file is up-to-date.

6.1.2 The CAMPUS Workload - Email

On CAMPUS, email is the dominant cause of traffic. During the peak load hours, about 20% of the unique files referenced are user inboxes, and another 50% are lock files used to control concurrent access to these inboxes. Many CAMPUS users use email applications that access additional configuration files, put incoming email into other mailboxes, and create temporary files to hold messages during composition. We do not identify each of these kinds of files here, but we have observed that a large fraction of the remaining accessed files are also related to email use.

These numbers and our analysis of the traces support the hypothesis that most CAMPUS users do little else on the system besides use email. A typical user session involves logging in (accessing `.cshrc` and possibly `.login`) and starting an email client. The email client typically reads a configuration file, creates a lockfile for the mailbox, and then scans the mailbox file. During a mail session, mail applications may rescan the mailbox several times. Composing email messages may, depending on the mail program, create temporary files in home directories, and viewing or extracting attachments may also create files. Quitting the mail client causes some or all of the mailbox file to be rewritten.

Even more dominant is the contribution of email to the total quantity of data movement. For both the total

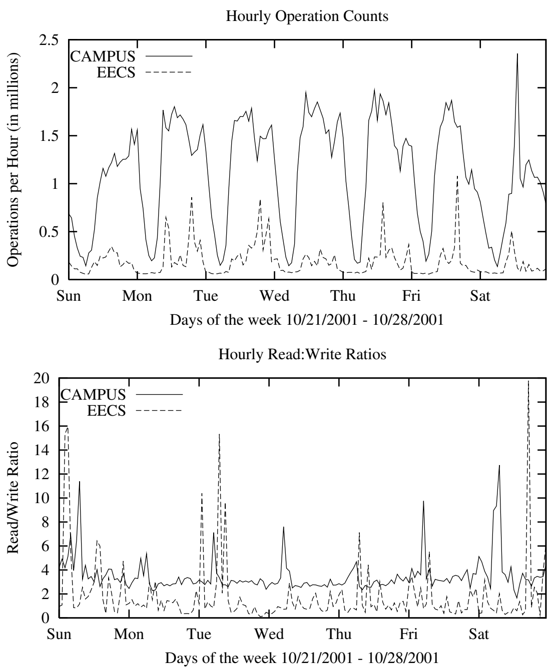

Figure 4: Variation of the hourly total operation count and read/write ratios for the week of 10/21/2001.

	All Hours			
	CAMPUS		EECS	
Total Ops (1000s)	1113	(48%)	185.1	(86%)
Data Read (MB)	4989	(45%)	212.3	(165%)
Read Ops (1000s)	719	(48%)	19.7	(110%)
Data Written (MB)	1856	(58%)	378.5	(246%)
Write Ops (1000s)	239	(58%)	28.6	(201%)
R/W Op Ratio	3.27	(48%)	3.16	(242%)
	Peak Hours Only			
	CAMPUS		EECS	
Total Ops (1000s)	1699	(7.6%)	267	(68%)
Data Read (MB)	7153	(6.1%)	268	(146%)
Read Ops (1000s)	1088	(7.1%)	29.2	(77%)
Data Written (MB)	2934	(12%)	439	(228%)
Write Ops (1000s)	377	(12%)	341	(158%)
R/W Op Ratio	2.46	(10%)	1.13	(106%)

Table 5: Average hourly activity. The *All Hours* columns are for the entire week of 10/21-10/27/2001. The peak hours are the hours 9am-6pm, Monday 10/22/2001 through Friday 10/26/2001. The numbers in parentheses are the standard deviations of the hourly averages, expressed as a percentage of the average.

number of read and write operations and the total volume of data transferred, more than 95% of the data read and written involve a user's primary inbox.

One of the causes of the large read load on CAMPUS is an unfortunate interaction between NFS's file-based caching model and the flat-file inbox and mail clients used on CAMPUS. Delivering a message to an inbox updates the modification time on the entire file, even though only a small number of blocks might actually have been changed. For a typical inbox on CAMPUS this results in the invalidation and immediate re-reading of, on average, more than 2 megabytes of data in the client cache. This component of the workload represents the majority of all reads on CAMPUS. We speculate that if client caching of mailboxes was done on a block or message basis instead of a file basis, the amount of data read per day would shrink to a fraction of the current size.

6.2 Variance of the Workloads due to Time

Like Vogels [15], we observe extremely large variance of load characterization statistics over time. We also observe, however, that much of this variance can be explained by high-level changes in the workload over time. This correlation has been observed in many trace studies, but its effects are usually ignored.

The most notable change in our traces is the difference between peak and off-peak hours, where peak hours are 9am-6pm on weekdays. Figure 4 illustrates the cyclical pattern of the CAMPUS load. It also portrays the consistent read/write ratio during peak hours and its tendency to spike during off-peak hours, when a few accesses can skew the ratio. Table 5 quantifies the reduction in normalized variance during peak hours, conveying that time is a strong predictor of operation counts, amount of data transferred, and the read-write ratio for CAMPUS.

We examined a range of possibilities for the "peak" hours for CAMPUS and found that using 9am-6pm resulted in the least variance. The standard deviation, expressed as a percentage of the mean, is reduced by a factor of at least 4 for all of the CAMPUS statistics when only these peak hours are considered. The same peak hours were also those that resulted in the least variance for EECS, although there is less correlation between the EECS workload and the "regular" work week. In many cases, the load spikes from Figure 4 are directly attributable to specific causes, such as software builds, large experiments, or data processing, which are often run via `cron` during off-hours.

Understanding the time-varying nature of a workload is essential for any kind of dynamically-optimizing storage system. Systems that experience genuine quiet periods can use them to rearrange data in anticipation of the next period of heavy use. Less radically, if the system is designed to make file layout decisions based upon workload, it is useful to know that there are atypical periods (for example, when backups are running) that might best be ignored, in order to avoid optimizing for an uncommon or non-critical workload.

6.3 Predicting File Attributes via File Names

One purpose of our data collection was to explore ways of predicting future file access patterns in order to optimize file and block layout. Cao *et al.* propose that applications can provide hints to the file system about what the application believes the future properties and access patterns of their files will be [4]. This method has not been widely adopted because it requires modifying the applications to provide these hints. However, we found that on CAMPUS and EECS, applications do provide hints in that the filenames chosen for new files are accurate predictors of the lifespan, size, and access patterns of nearly all files. We also observe that file renames are rare, which means that these hints are rarely wrong or need to be modified. This means that the file system has, at the time of file creation, reliable and potentially useful information to guide its decisions.

On CAMPUS we can predict the size, lifespan, and access patterns of most files extremely well simply by examining the last component of the pathname. Nearly all of the files on CAMPUS fall into one of the four categories: lock files, dot files, mail composer files, and mailboxes, and the size, lifespan, and access patterns are predicted strongly for each of these categories.

Zero-length lock files make up 96% of the files that are both created and deleted during our test week, and 99.9% of these lock files live less than 0.40 seconds. Temporary files created by the mail composer account for 2.5% of the files created each day; 45% of these live less than 1 minute, 98% are less than 8K in length, and 99.9% are smaller than 40K. Most dot files fit in one block, although there are some multi-block dot files (for example, the primary mail client configuration file `.pinerc` varies in size from 11K to 26K). The mailbox files are considerably larger than any other commonly-accessed file and are never deleted.

Activity on EECS is a union of several kinds of activities, and the combined workload is more complex than CAMPUS. However, our preliminary analyses show that for most files on EECS, the pathname of a file is also a strong predictor of file attributes. Analyzing these relationships and developing methods for the file system to infer and exploit them is the subject of ongoing work.

6.4 Run Patterns and Sequentiality

In order to quantify the degree of sequentiality in a run to a finer level than simply sequential or random, we introduce a *sequentiality metric*, based on Keith Smith's *layout score* [14]. Our *sequentiality metric* is the fraction of blocks that have been accessed sequentially. A block is accessed sequentially if it is consecutive to the previous access. A run with a sequentiality metric close to 1.0 is almost sequential and should be processed by the file system and disk as if it were.

To account for the minimal penalty introduced by small jumps, we introduce the term *δ-consecutive* to mean that a block is within δ blocks of its predecessor. In our analyses we have arbitrarily chosen a δ of 10 blocks; logical jumps of less than 10 blocks are unlikely to require seeks, but these jumps also account for a large percentage of the jumps we observed.

Figure 5 examines the differences in average sequentiality metrics for different run lengths. Long reads on CAMPUS are typically highly sequential. Long CAMPUS writes, however, tend to touch several sequential blocks and then seek to a new location, either forward or backward in the file. This is reflected in a metric of about 0.6, which means that only 60% of the accesses are δ-consecutive. Long EECS reads also tend to exhibit highly sequential behavior, although not to as great a degree as CAMPUS. The jaggedness of the plot of long EECS runs is caused by the scarcity of occurrences of runs of this size in EECS and the high variance between those occurrences. EECS writes still tend to be highly-seek prone, often having average sequentiality metrics below 0.5, even when $\delta = 10$.

The central difficulty in the results produced through the mechanisms of partial reordering (described in Section 4.2) and allowing small jumps is that they only enable approximations of the clients' behaviors. If a series of requests is genuinely out-of-order or makes small jumps, these two mechanisms will obscure this behavior. However, an NFS server must take into account client activity when deciding to prefetch. That requests are reordered despite the best efforts at the client to issue them in an optimal order suggests that a more intelligent algorithm must be used by the server to optimize accesses.

To investigate the effect of reordered requests on NFS read performance, we performed an experiment by modifying the FreeBSD 4.4 NFS server to employ a simplified version of the sequentiality metric presented here in its read-ahead heuristic. On a loaded system, we observed that nearly 10% of the requests were reordered, and in this situation our new heuristic improved end-to-end transfer speed for large sequential transfers by more than 5%.

The need to apply heuristics to adequately support client-side access patterns and the results of our experiment suggest that NFS servers must be intelligent in categorizing a client's access: a single out-of-order access should not relegate it to the "random" dustbin. In our traces, the vast majority of seeks were to blocks two or three away from the current offset. These may have been due to lost packets or may be a genuine reflection of the

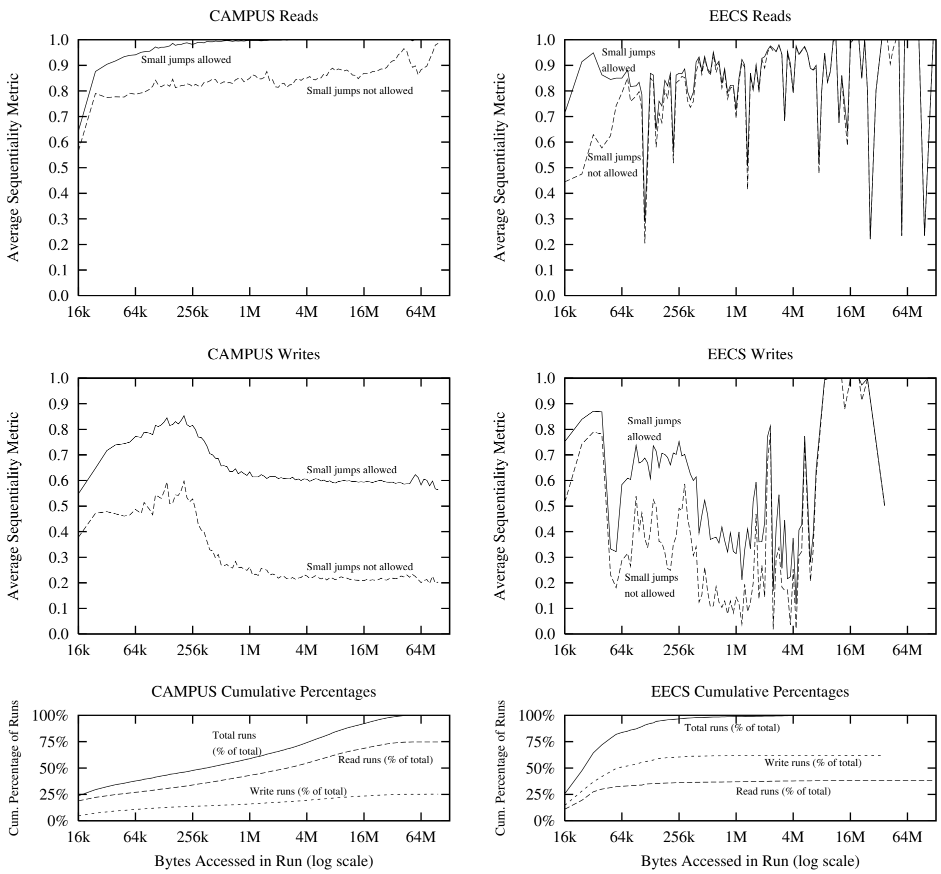

Figure 5: Bytes Accessed vs. Sequentiality Metric. $\delta = 10$ shown as "small jumps allowed"; $\delta = 0$ as "small jumps not allowed."

access stream, but in either case runs containing only adjacent and nearby requests are served more efficiently if they are considered sequential. In our categorization, we consider any jump of fewer than 10 blocks sequential, because on today's disks, if the file is laid out contiguously on disk, then logical seeks of fewer than 10 blocks are unlikely to induce disk arm movement.

7 Conclusion

Some of our results strengthen or support the heuristics behind contemporary file system design, while others portray new aspects of NFS workloads that should be taken into consideration in the design of future file systems and NFS servers. Our analysis of our new traces also suggests several new research questions.

- We find that NFS servers must be prepared to deal with reordered calls. If they use fragile metrics to estimate the sequentiality of an access pattern, reordered calls may degrade their performance.
- Optimizations based on file pathnames could assist servers in optimizing their file layout. The predictions are not uniform across systems, but must be learned from observations of each system. The predictions are particularly accurate on single-application systems, such as mail servers, which are becoming more common.
- Our traces show a continuation in the trend that in traditional computer science workloads, many blocks die quickly. We also see this in email workloads. Mechanisms for delaying writes, such as NVRAM, would improve performance for both the CAMPUS and EECS workloads.
- Most long read runs that conventional metrics label as "random" actually contain long, entirely sequential sub-runs, so servers that recognize these somewhat more complex patterns can improve performance by optimizing the files for sequential access.
- Although long write runs exhibit higher variance in sequentiality than long read runs, 60% of their block accesses are sequential.
- On EECS and especially CAMPUS, servers could schedule periods of reorganization since the daily

and weekly pattern of the workload is predictable.

- While it may have been obvious *a priori*, flat-file mailboxes are quite inefficient. Anecdotal evidence and recent experiments suggest that database-driven mail servers can be faster and consume fewer resources than file-system based servers [6]. It might be possible to build a file system to serve mail loads as efficiently as database-driven mail servers, but it is not clear whether such a file system offers any compelling advantages.

Our findings suggest a number of ways that file servers can optimize their performance by analyzing the workload they observe, but it remains to be seen whether the benefit of those optimizations is worth the cost. For example, we do not know how much data and computation are necessary for a general purpose file system to derive and take advantage of the strong correlation between filenames and file size or lifespan. The larger question is whether it makes sense to attempt this optimization at all, or instead simply accept the fact that general purpose file systems can never perform as well as specialized ones, and focus our efforts on designing file systems for specific workloads such as mail service, WWW service, the desktop, and other new workloads as they emerge.

8 Acknowledgments

We could not have collected our traces without generous help from the administrators of each system, particularly Alan Sundell and Bill Ouchark of CAMPUS and Peg Schaefer and Aaron Mandel of EECS. The paper benefited enormously from the thoughtful comments from our reviewers and Remzi Arpaci-Dusseau, our paper shepherd. This work was funded in part by IBM.

9 Obtaining a Copy of Our Traces

Researchers interested in acquiring a copy of our tools for collecting anonymized traces, or a copy of the anonymized copies of the traces described in this paper (or newer traces, as they become available) should contact the authors at *sos@eecs.harvard.edu*. We are in the process of constructing a public repository for traces and related tools, and are actively gathering contributions from other researchers.

References

[1] Mary G. Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John K. Ousterhout. Measurements of a Distributed File System. In *Proceedings of 13th ACM Symposium on Operating Systems Principles*, pages 198–212, Monterey, CA, October 1991.

[2] Matthew A. Blaze. NFS Tracing by Passive Network Monitoring. In *Proceedings of the USENIX Winter 1992 Technical Conference*, pages 333–343, San Fransisco, CA, January 1992.

[3] Matthew A. Blaze. *Caching in Large-Scale Distributed File Systems*. PhD thesis, Princeton University, January 1993.

[4] Pei Cao, Edward W. Felten, and Kai Li. Implementation and Performance of Application-Controlled File Caching. In *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 165–177, Monterey, CA, November 1994.

[5] Michael Dahlin, Randolph Wang, Thomas E. Anderson, and David A. Patterson. Cooperative Caching: Using Remote Client Memory to Improve File System Performance. In *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 267–280, Monterey, CA, 1994.

[6] Nicholas Elprin and Bryan Parno. An Analysis of Database-Driven Mail Servers. Technical Report TR-02-13, Harvard University DEAS, 2002.

[7] Riccardo Gusella. The Analysis of Diskless Workstation Traffic on an Ethernet. Technical Report 379, University of California at Berkeley, 1987.

[8] Andrew W. Moore. Operating System and File System Monitoring: a Comparison of Passive Network Monitoring with Full Kernel Instrumentation Techniques. Master's thesis, Monash University, 1995.

[9] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A Trace-Driven Analysis of the UNIX 4.2 BSD File System. In *Proceedings of the Tenth ACM Symposium on Operating Systems Principles*, pages 15–24, Orcas Island, WA, December 1985.

[10] B. Pawlowski, S. Shepler, C. Beame, B. Callaghan, M. Eisler, D. Noveck, D. Robinson, , and R. Thurlow. The NFS Version 4 Protocol. In *Proceedings of the 2nd International System Administration and Networking Conference (SANE2000)*, Maastricht, The Netherlands, May 2000.

[11] S. Quinlan and S. Dorward. Venti: a New Approach to Archival Storage. In *First USENIX Conference on File and Storage Technologies*, pages 89–102, Monterey, CA, 2002.

[12] Drew Roselli, Jacob Lorch, and Thomas Anderson. A Comparison of File System Workloads. In *USENIX 2000 Technical Conference*, pages 41–54, San Diego, CA, 2000.

[13] Mendel Rosenblum and John K. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10(1):26–52, 1992.

[14] Keith A. Smith and Margo I. Seltzer. File System Aging - Increasing the Relevance of File System Benchmarks. In *Proceedings of SIGMETRICS 1997: Measurement and Modeling of Computer Systems*, pages 203–213, Seattle, WA, June 1997.

[15] Werner Vogels. File System Usage in Windows NT. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, pages 93–109, Kiawah Island, SC, December 1999.